


C# Avançado

Operadores e Casting




Tópicos Abordados



- Uso de *checked* e *unchecked*
- Sobrecarga de operadores
- Definição de indexadores
- Casting customizados

Relembrando o Casting



- Casting é uma operação que permite alterar o tipo de um dado

```
int i = 10;  
long l = i;
```

int → long

Casting implícito


```
double d = 3.5;  
int i = (int)d;
```

double → int

Casting explícito

Pode ocorrer perda de informação

A Palavra-Chave *checked*



- Este código executa com sucesso

```
byte b1 = 100;
byte b2 = 200;
byte soma = (byte) (b1 + b2);
```

Qual o valor de *soma*? 44


O tipo *byte* só comporta valores de 0 a 255

- O uso do *checked* verifica esse “estouro” (overflow)

```
checked
{
    byte soma = (byte) (b1 + b2);
}
```


Lança uma exceção do tipo *System.OverflowException*

Habilitando a Checagem no Compilador



- O compilador do C# dá a opção de habilitar a mesma checagem que o *checked* faz
 - A checagem vai ocorrer no código todo
- Usar a opção **/checked**
 - O Visual Studio permite habilitar esta checagem nas configurações do projeto
- O ideal é usar esta configuração apenas durante o desenvolvimento e testes
 - Prejudica a performance da aplicação

A Palavra-Chave *unchecked*



- Quando a checagem de overflow está ativada no compilador, o *unchecked* pode ser utilizado para permitir o overflow

```
unchecked
{
    byte soma = (byte) (b1 + b2);
}
```

A exceção do tipo não é mais lançada

Sobrecarga de Operadores



- Permite que uma classe/estrutura sobrescreva o comportamento de um operador
- Exemplo: classe *string*

```
string s1 = "Linguagem";  
string s2 = "C#";  
string s3 = s1 + " " + s2;
```

O operador "+"
concatena duas strings

```
string s1 = "ABC";  
string s2 = "ABC";  
  
if (s1 == s2)  
{  
    //...  
}
```

O operador "==" testa a
igualdade do conteúdo
dos objetos *string*

Como Sobrescrever Operadores



```
class Quadrado  
{  
    double lado;  
  
    public Quadrado(double lado)  
    {  
        this.lado = lado;  
    }  
}
```

```
Quadrado q1 = new Quadrado(5);  
Quadrado q2 = new Quadrado(10);  
  
if (q1 < q2) {}  
if (q1 > q2) {}
```

Comparar dois objetos
Quadrado com os
operadores "<" e ">"

Sobrecarregar os operadores

```
public static bool operator >(Quadrado q1, Quadrado q2)  
{  
    return q1.lado > q2.lado;  
}  
  
public static bool operator <(Quadrado q1, Quadrado q2)  
{  
    return q1.lado < q2.lado;  
}
```

Mais um Exemplo



```
Quadrado q = new Quadrado(10);  
q++;
```

Incrementar um
objeto *Quadrado*

```
public static Quadrado operator ++(Quadrado q)  
{  
    return new Quadrado(q.lado + 1);  
}
```

Use a sobrecarga com cuidado, a fim de
evitar a sobrecarga de operadores que não
fazem muito sentido

Operadores Sobrecarregáveis



- Os principais operadores que podem ser sobrecarregados são:

Tipo de Operadores	Operadores
Unários	!, ++, --
Binários	+, -, *, /, %
Relacionais	==, !=, <, >, <=, >=

Definindo Indexadores



- O operador de indexação é bastante utilizado com arrays

```
double[] notas = new double[3];  
notas[0] = 7.5;  
notas[1] = 6.3;  
notas[2] = 9.7;
```

```
double nota = notas[1];
```

- É possível definir este operador em uma classe ou estrutura

Definindo Indexadores



```
class Notas  
{  
    double nota1;  
    double nota2;  
    double nota3;  
  
    public double this[int index]  
    {  
        get  
        {  
            switch (index)  
            {  
                case 0: return nota1;  
                case 1: return nota2;  
                case 2: return nota3;  
                default: return -1; }  
        }  
  
        set  
        {  
            switch (index)  
            {  
                case 0: nota1 = value; break;  
                case 1: nota2 = value; break;  
                case 2: nota3 = value; break; }  
        }  
    }  
}
```

Sintaxe semelhante à
definição de uma property

O bloco set tem acesso à
variável value

```
Notas n = new Notas();  
n[0] = 7.5;  
n[1] = 6.3;  
n[2] = 9.7;
```

```
double nota = n[1];
```

Tipo de Dado do Indexador



- No exemplo anterior, o indexador era um número inteiro
- Isto não é obrigatório
 - Qualquer tipo de dado pode ser usado

```
class Pessoas
{
    public Pessoa this[string nome]
    {
        ...
    }
}
```

Indexador do tipo *string*

```
Pessoas pessoas = new Pessoas();
Pessoa p1 = p["José"];
Pessoa p2 = p["Julia"];
```

Múltiplos Indexadores



- Uma mesma classe/estrutura pode definir múltiplos indexadores
 - Desde que os parâmetros sejam de tipos diferentes e/ou em quantidades diferentes

```
public MinhaClasse this[string s] { }
public MinhaClasse this[int i] { }
public int this[string s, bool b] { }
```

Indexador de mais de uma dimensão

```
int x = p["abc", true];
```

Casting Customizado



- A linguagem C# permite que o programador implemente casting customizado para uma classe ou estrutura

```
class Quadrado
{
    public int Lado { get; set; }

    public static explicit operator int(Quadrado q)
    {
        return q.Lado;
    }

    public static implicit operator Quadrado(int l)
    {
        return new Quadrado() { Lado = l };
    }
}
```

Casting explícito de *Quadrado* para *int*

Casting implícito de *int* para *Quadrado*

Casting Customizado



```
class Quadrado
{
    public int Lado { get; set; }

    public static explicit operator int(Quadrado q) {
        return q.Lado;
    }

    public static implicit operator Quadrado(int l)
    {
        return new Quadrado() { Lado = l };
    }
}
```

Quadrado q = new Quadrado() { Lado = 10 };
int l = (int)q;

l = 10

Quadrado q = 5;
int l = q.Lado;

l = 5



Softblue
